Dependency Parsing with Energy-based Reinforcement Learning

Lidan Zhang
Department of Computer Science
The University of Hong Kong
Pokfulam Road, Hong Kong
lzhang@cs.hku.hk

Kwok Ping Chan
Department of Computer Science
The University of Hong Kong
Pokfulam Road, Hong Kong
kpchan@cs.hku.hk

Abstract

We present a model which integrates dependency parsing with reinforcement learning based on Markov decision process. At each time step, a transition is picked up to construct the dependency tree in terms of the long-run reward. The optimal policy for choosing transitions can be found with the SARSA algorithm. In SARSA, an approximation of the state-action function can be obtained by calculating the negative free energies for the Restricted Boltzmann Machine. The experimental results on CoNLL-X multilingual data show that the proposed model achieves comparable results with the current state-of-the-art methods.

1 Introduction

Dependency parsing, an important task, can be used to facilitate some natural language applications. Given a sentence, dependency parsing is to find an acyclic labeled directed tree, projective or non-projective.The label of each edge gives the syntactic relationship between two words.

Data-driven dependency parsers can be categorized into graph-based and transition-based models. Both of these two models have their advantages as well as drawbacks. As discussed in (McDonald and Satta, 2007), transition-based models use local training and greedy inference algorithms, with a rich feature set, whereas they might lead to error propagation. In contrast, graph-based models are globally trained coupled with exact inference algorithms, whereas their features are restricted to a limited number of graph arcs. Nivre and McDonald (2008) presented a successful attempt to integrate these two models by exploiting their complementary strengths.

There are other researches on improving the individual model with a novel framework. For example, Daumé et al. (2006) applied a greedy search to transition-based model, which was adjusted by the resulting errors. Motivated by his work, our transition-based model is expected to overcome local dependencies by using a long-term desirability introduced by reinforcement learning (RL). We rely on a "global" policy to guide each action selection for a particular state during parsing. This policy considers not only the current configuration but also a few of look-ahead steps. Thus it yields an optimal action from the long-term goal. For example, an action might return a high value even if it produces a low immediate reward, because its following state-actions might yield high rewards. The reverse also holds true. Finally we formulate the parsing problem with the Markov Decision Process (MDP) for the dynamic settings.

The reminder of this paper is organized as follows: Section 2 describes the transition-based dependency parsing. Section 3 presents the proposed reinforcement learning model. Section 4 gives the experimental results. Finally, Section 5 concludes the paper.

2 Transition-based Dependency Parsing

In this paper, we focus on the transition-based dependency parsing in a shift-reduce framework (Kübler et al., 2009). Given a sentence $x = w_0, w_1, ..., w_n$, its dependency tree is constructed by a sequence of transitions. The data structures include a stack S to store partially processed words and a queue I to record the remaining input words and the partial labeled dependency structure constructed by the previous transitions. Four permissible transitions are considered: **Reduce**: pops word w_i from the stack; **Shift**: pushes the next input w_j onto the stack; **Left-Arc**$_r$: adds a labeled dependency arc r from the next input w_j to the top of the stack w_i, then pops word w_i from the stack; **Right-Arc**$_r$: adds a dependency arc r

Proceedings of the 11th International Conference on Parsing Technologies (IWPT), pages 234–237,
Paris, October 2009.

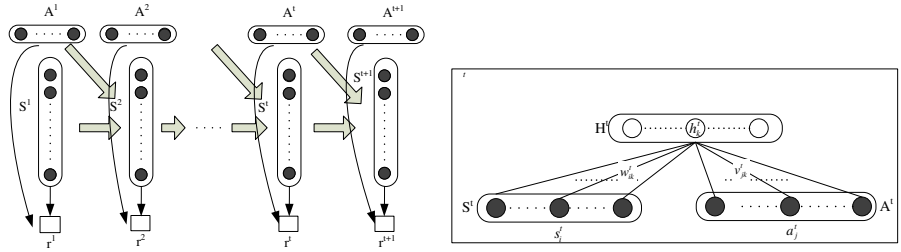

Figure 1: The MDP with factored states and actions. Left: The general network. Right: Detailed network with one hidden layer at time t. Visible variables (states and actions) are shaded. Clear circles represent hidden variables.

from the top of the stack w_i to the next input w_j, and pushes word w_j onto the stack.

Starting from the empty stack and initializing the queue I as the input words, the parser terminates when the queue I is empty. The optimal transition (or say, action/decision A) in each step is conditioned on the current configuration c of the parser. For non-projective cases, preprocessing and postprocessing are applied.

3 Reinforcement Learning

3.1 General Framework

We begin with looking at the general framework to integrate RL into the transition-based dependency model. In this paper, we reformulate the dependency parsing as Markov Decision Process (MDP, $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r)$) where:

- $\mathcal{S}$ is the set of states.
- $\mathcal{A}$ is the set of possible actions.
- $\mathcal{T}$ is the transition function, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. we denote the transition probability $P_{ij}(a) = P(s^{t+1} = j | s^t = i, A^t = a)$.
- r is the reward function by executing action a in a certain state, which is denoted as $r_i(a)$.

As aforesaid, the key task of dependency parsing is to select the optimal action to be performed based on the current state. Given the expected immediate reward r, the optimal policy ($\pi : S \mapsto A$) is to maximize the long-term expected reward as follows:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{1}$$

Given a policy π, state-action function $Q^\pi(i, a)$ can be defined as the expected accumulative reward received by taking action a in state s. It takes the following form:

$$\begin{aligned} Q^\pi(i,a) &= E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s^t = i, a^t = a] \\ &= \sum_j P_{ij}(a)[r_i(a) + \gamma \sum_b \pi(j,b) Q^\pi(j,b)] \end{aligned} \tag{2}$$

Here $\pi(j, b)$ is the probability of picking up action b in state j, $\gamma \in [0, 1]$ is a discount factor to control the involvement of further actions. According to the Bellman equation, the state-action function can be updated iteratively with equation(2).

Given the state-action function, a greedy policy can be found by maximizing over possible actions:

$$\pi' = \arg\max_a Q^\pi(i, a) \tag{3}$$

In the following, we will discuss how to compute the state-action function Q by investigating the free energy in RBM.

3.2 Restricted Boltzmann Machine

3.2.1 Free Energy

Figure 1 shows the general framework of our model. At each time step t, there is no connections between nodes within the same layer. In the network, "visible" variables include both states and actions ($V = S \cup A$). The visible layer is fully connected to a "hidden" layer, which can be regarded as a Restricted Boltzmann Machine (RBM).

In our model, both states and actions are factored. They are consisted of a sets of discrete variables (Sallans and Hinton, 2004). The stochastic energy of the network can be computed by the conductivities between visible and hidden variables.

$$E(s, a, h) = -\sum_{i,k} w_{ik} s_i h_k - \sum_{j,k} \mu_{jk} a_j h_k \tag{4}$$

The above energy determine their equilibrium probabilities via the Boltzmann distribution:

$$P(s,a,h) = \frac{exp(-E(s,a,h))}{\sum_{s',a',h'} exp(-E(s',a',h'))} \quad (5)$$

By marginalizing out the hidden variables, we can obtain the "equilibrium free energy" of s and a, which can be expressed as an expected energy minus an entropy:

$$\begin{aligned} F(s,a) = & -\sum_k (\sum_i (w_{ik} s_i \langle h_k \rangle) + \sum_j (\mu_{jk} a_j \langle h_k \rangle)) \\ & + \sum_k \langle h_k \rangle \log \langle h_k \rangle + (1 - \langle h_k \rangle) \log (1 - \langle h_k \rangle) \end{aligned} \quad (6)$$

where $\langle h_k \rangle$ is the expected value of variable h_k:

$$\langle h_k \rangle = \sigma(\sum_{i,k} w_{ik} s_i + \sum_{j,k} \mu_{jk} a_j) \quad (7)$$

where $\sigma = 1/(1+e^{-x})$ is a sigmoid function.

As is proved in (Sallans and Hinton, 2004), the value of a state-action function can be approximated by the negative free energy of the network:

$$Q(s,a) \approx -F(s,a) \quad (8)$$

3.2.2 Parameter Learning

The parameters of the network can be updated by the SARSA (State-Action-Reward-State-Action) algorithm. The inputs of the SARSA algorithm are the state-action pairs of the two neighboring slices. Then the error can be computed as:

$$\mathcal{E}(s^t, a^t) = [r^t + \gamma Q(s^{t+1}, t^{t+1})] - Q(s^t, a^t) \quad (9)$$

Suppose the state-action function is parameterized by θ. The update equation for the parameter is:

$$\triangle\theta \propto \mathcal{E}(s^t, a^t) \nabla_\theta Q(s^t, a^t) \quad (10)$$

Back to our model, the parameters $\theta = (w, u)$ are given by:

$$\begin{aligned} \Delta w_{ik} &\propto (r^t + \gamma Q(s^{t+1}, a^{t+1}) - Q(s^t, a^t)) s_i^t \langle h_k \rangle \\ \Delta u_{jk} &\propto (r^t + \gamma Q(s^{t+1}, a^{t+1}) - Q(s^t, a^t)) a_j^t \langle h_k \rangle \end{aligned} \quad (11)$$

Leemon (1993) showed that the above update rules can work well in practice even though there is no proof of convergence in theory. In addition, in dependency parsing task, the possible action number is small (=4). Our experimental results also showed that the learning rule can converge in practice.

3.3 Action Selection

After training, we use the *softmax* rules to select the optimal action for a given state. The probability of an action is given by Boltzmann distribution:

$$P(a|s) \approx \frac{e^{Q(s,a)/\tau}}{Z} \quad (12)$$

Here Z is an normalization factor. τ is a positive number called the *temperature*. High temperature means the actions are evenly distributed. Low temperature case a great variety in selection probability. In the limit as $\tau \to 0$, softmax action selection becomes greedy action selection.

4 Experiments

4.1 Settings

We use the CoNLL-X (Buchholz and Marsi, 2006) distribution data from seven different languages (Arabic, Bulgarian, Dutch, Portuguese, Slovene, Spanish and Swedish). These treebanks varied in sizes from 29,000 to 207,000 tokens. The cut-off frequency for training data is 20, which means we ignores any attribute (FORM, LEMMA, POS or FEATS) occurred less than 20. Furthermore we randomly selected 10 percent of training data to construct the validation set. Test sets are about equal for all languages. Since our algorithm only deals with projective cases, we use projectivization/deprojectivization method for training and testing data.

For fair comparison, we use the exactly same feature set as Nivre et al. (2006), which is comprised of a variety of features extracted from the stack, the queue and the partially built dependency graph.

In our experiment, the immediate reward value is defined as the Hamming Loss between partial tree and expected tree, which counts the number of places that the partial output $\hat{y}$ differs from the true output y: $\sum_{i=1}^{T} 1[y_i \neq \hat{y}_i]$.

As shown in Figure 1, we compute the state-action function using a feed-forward neural network with one hidden layer. The number of hidden variables is set to match the variable number in the visible layer (i.e. total number of state and action variables). The parameters of the network are modified by SARSA algorithm according to equation 2. Finally, 10-width beam search is employed for all languages, during testing.

There are other parameters in our experiments, which can be tuned using search. For simplicity,

		Ar	Bu	Du	Po	Sl	Sp	Sw
LAS	Our	63.24	88.89	79.06	87.54	72.44	82.79	87.20
	Nivre	66.71	87.41	78.59	87.60	70.30	81.29	84.58
UAS	Our	75.30	92.88	83.14	91.34	80.06	86.18	91.84
	Nivre	77.51	91.72	81.35	91.22	78.72	84.67	89.50

Table 1: Comparison of dependency accuracy with Nivre

the learning rate was exponentially decreased form 0.1 to 0.01 in the course of each epoch. In ideal cases, the discount factor should be set to 1. In our experiments, discount factor is fixed to 0.6 considering the computational burden in long sentence. The study of the this parameter is still left for future work. Finally, the inverse temperature linearly increased from 0 to 2.

4.2 Results

The performance of our model is evaluated by the official attachment score, including labeled (LAS=the percentage of tokens with the correct head and label) and unlabeled (UAS=the percentage of tokens with the correct head). Punctuation tokens were excluded from scoring.

The result comparison between our system and Nivre's transition-based system is shown in Table 1[1]. From the table, we can see that the proposed model outperformed the Nivre's score in all languages except Arabic. In Arabic, our results are worse than Nivre, with about 3.5% performance reduction in LAS measure and 2.2% in UAS. Most of our errors occur in POSTAGS with N (16% head errors and 31% dep errors) and P (47% head errors and 8% dep errors), which is probably due to the flexible usage of those two tags in Arabic. The best performance of our model happens in Swedish. The LAS improves from 84.58% to 87.20%, whereas UAS improves from 89.5% to 91.84%. The reason might be that the long dependent relationship is not popular in Swedish. Finally, we believe the performance will be further improved by carefully tuning parameters or broadening the beam search width.

5 Conclusions

In this paper we proposed a dependency parsing based on reinforcement learning. The parser uses a policy to select the optimal transition in each parsing stage. The policy is learned from RL in terms of the long-term reward. Tentative experimental evaluations show that the introduction of RL is feasible for some NLP applications. Finally, there are a lot of future work, including the hierarchical model and parameter selections.

[1]The performance of other systems can be accessed from http://nextens.uvt.nl/∼conll

References

Sabine Buchholz and Erwin Marsi. 2006. Conll-x shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, New York City, June. Association for Computational Linguistics.

Hal Daumé III, John Langford, and Daniel Marcu. 2006. Searn in practice.

Leemon C. Baird III and A. Harry. Klopf. 1993. Reinforcement learning with high-dimensional, continuous actions. Technical Report WL–TR-93-1147, Wright-Patterson Air Force Base Ohio: Wright Laboratory.

Sandra Kübler, Ryan McDonald, and Joakim Nivre. 2009. *Dependency parsing*. Calif, Morgan & Claypool publishers, US.

Ryan McDonald and Giorgio Satta. 2007. On the complexity of non-projective data-driven dependency parsing. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 121–132, Prague, Czech Republic, June. Association for Computational Linguistics.

Joakim Nivre and Ryan McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *Proceedings of ACL-08: HLT*, pages 950–958, Columbus, Ohio, June. Association for Computational Linguistics.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülşen Eryiğit, and Svetoslav Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 221–225, New York City, June. Association for Computational Linguistics.

Brian Sallans and Geoffrey E. Hinton. 2004. Reinforcement learning with factored states and actions. *Journal of Machine Learning Research*, 5:1063–1088.